Journal of Parallel and Distributed Computing, Volume 141, July 2020, Pages 82-98.

# Cost Effective Dynamic Data Placement for Efficient Access of Social Networks

Hourieh Khalajzadeh[1], Dong Yuan[2], Bing Bing Zhou[3], John Grundy[1], Yun Yang[4]

[1]*Faculty of Information Technology, Monash University, VIC 3168, Australia.*
*E-mail: {hourieh.khalajzadeh, john.grundy}@monash.edu. Phone: +61 3 9905 6968, +61 3 9905 8854*
[2]*School of Electrical and Information Engineering, the University of Sydney, NSW 2006, Australia.*
*E-mail: dong.yuan@sydney.edu.au. Phone: +61 2 8627 2007*
[3]*School of Information Technologies, the University of Sydney, NSW 2006, Australia.*
*E-mail: bing.zhou@sydney.edu.au. Phone: +61 2 9036 9112*
[4]*School of Software and Electrical Engineering, Swinburne University of Technology, VIC 3122, Australia.*
*E-mail: yyang@swin.edu.au. Phone: +61 3 9214 8752*

Abstract—Social networks boast a huge number of worldwide users who join, connect, and publish various content, often very large, e.g. videos, images etc. For such very large-scale data storage, data replication using geo-distributed cloud services with virtually unlimited capabilities are suitable to fulfill the users' expectations, such as low latency when accessing their and their friends' data. However, service providers ideally want to spend as little as possible on replicating users' data. Moreover, social networks have a dynamic nature and thus replicas need to be adaptable according to the environment, users' behaviors, social network topology, and workload at runtime. Hence, it is not only crucial to have an optimized data placement and request distribution – meeting individual users' acceptable latency requirements while incurring minimum cost for service providers – but the data placement must be adapted based on changes in the social network to keep it efficient and effective over time. In this paper, we model data placement as a dynamic set cover problem and propose a novel approach to solve this problem. We have run several experiments using two large-scale, open Facebook and Gowala datasets and real latencies derived from Amazon cloud datacenters to demonstrate our novel strategy's efficiency and effectiveness.
**Index Terms**— Access latency, cloud computing, cost optimization, data placement, data replication, greedy algorithm, set cover problem, social networks

## 1 Introduction

Based on a recent report, there were 2.80 billion global social media users in 2017, or roughly 37% of the population of the world, with more than 20% growth over the previous 12 months [1]. Users are geographically scattered around the world often having friendships with users from elsewhere. Participating users join a social network, create friendships with any other users with whom they associate, and publish various content – some such as videos and images being very large – to share with each other. Facebook (2 billion monthly active users), YouTube (1.5 billion monthly active users), WhatsApp (1.2 billion monthly active users), and Instagram (700 million monthly active users) are some examples of popular social networks [2] with large social media data content. Social network users have quality of service expectations from their social network service provider, including low latency, data consistency and availability, and privacy requirements. In terms of latency, users can endure a certain threshold to access their own data and the data of their friends. Not being able to access the data in a desirable timeframe is likely to lead users to becoming frustrated, lowering their usage and possibly even leaving the social network.
Replication of data can be utilized to meet these user performance requirements. Geo-distributed cloud services with virtually unlimited capabilities are suitable for such large-scale data storage. There are many cloud service providers maintaining storage infrastructure based on a pay-as-you-go model [3-5]. Amazon S3 [6], Google Cloud storage [7], and Microsoft Azure [8] are some examples. However,

such cloud rental is extremely costly when using naïve full replication for social media data in order to minimize the latency, ensure availability and meet other requirements. Moreover, as different replicas of users' data may need periodical synchronization, such a huge investment becomes uneconomical due to the very large size of datasets. Based on Facebook statistics for 2016 [9], Facebook generates four petabytes of new data per day for the 1.083 billion daily active users at that time.
Social networks have a dynamic and growing nature. Users can join or leave the network at any time; add, delete or update datasets; travel and move to another location in the world; and also create or break friendships. Different users have different levels of activeness and similarly their friends have various frequencies of accessing, updating and adding to their data. Users can become less or more active and gain less or more interest in their friends' shared contents. Finally, sometimes new datacenters can be added for use by the social network or occasionally the existing datacenters can be out of use, hence removed.
An optimized data placement approach is required that must be capable of finding the most cost effective solution while fulfilling individual users' acceptable latency requirement. Furthermore, the data placement needs to be able to cope with the dynamic changes in the environment, e.g., users' behaviors, social network topology, and workload at runtime. Hence**, finding the optimal placement of data in different datacenters with minimum cost while keeping the placement optimized over time is the key challenge addressed in this paper**.
We aim to guarantee the latency requirement for $P^{\text{th}}$ percentile of *all individual requests* between all friends with a minimum cost, i.e., over $P\%$ of all individual operations are within specified latency requirement.
We present a novel dynamic strategy to cope with data placement that is applicable in dynamic environments where users can join, leave, move or change their friendships in the social network; data can be added, removed and updated as needed; and datacenters can also be added or removed. To the best of our knowledge, our work is the only comprehensive work dedicated to all different scenarios that happen in a social network for data placement. Four significant contributions of this research are:

- Finding the initial minimum number of replicas for every user and relay different requests to the best datacenters in order to ensure the latency requirement.
- Presenting a novel dynamic strategy that continuously guarantees the optimality of the social network data placement over time. Our dynamic strategy is based on our static minimum cost replication strategy in order to make it practical in the real world where social networks change rapidly.
- Fulfilling $P^{\text{th}}$ percentile requirement of individual access latencies of all users. Taking individual instead of average latencies into account makes our work much more practical and significantly distinct from other existing works.
- Guaranteeing a very low – almost unnoticeable – individual latency of less than 250 ms (milliseconds) [10] not only for users to access their own data but also for all their friends to access their data.

We model and map the complex problem of finding a cost effective data placement strategy and coping with the changes in the social network while fulfilling the latency requirement for individual requests to the simple well-known (dynamic) set cover problem in order to solve it efficiently and effectively. To derive the most cost effective solution, a framework consisting of a combination of greedy and dynamic greedy algorithms is used to find the most affordable solution. A greedy algorithm - one of the most effective heuristic algorithms to solve the set cover problem [11] - is used to find the minimum number of appropriate datacenters for every user that, by replicating data in them, it is possible to have the latency requirement fulfilled for this user's friends. Our overall goal is to continuously adapt the data placement for a given set of users' data replicas based on the changes to the social network so that we have the minimum storage, transfer and synchronization cost while guaranteeing that $P^{\text{th}}$ percentile of individual latencies is no more than the acceptable latency. Real Amazon datacenters are tested to use real-world datacenter latency measurements. Two large-scale open datasets, Facebook dataset [12] and location based Gowala dataset [13], are used to evaluate our novel strategy and demonstrate its efficiency and effectiveness. As verified by simulation experiments, our dynamic strategy to solve the data placement as a dynamic set cover problem is capable of finding the optimal solution.
Section 2 provides a motivating example of social network data placement with problem analysis. Section 3 introduces our set cover based problem formulation. Section 4 presents our dynamic framework including the greedy and dynamic greedy algorithms in more detail. Section 5 demonstrates the simulation results and the evaluation. Section 6 discusses related work. Finally, Section 7 summarizes conclusions and key areas for future research.

## 2 Motivating Example and Problem Analysis

Social networks deal with an extremely large number of users distributed all around the world sharing

a rapidly growing volume of interconnected and often large data items. Users typically have friends in diverse places who expect to access their data promptly, i.e., with a small latency.
For example, let us consider a user who has active friends in North America, India, Europe and Australia. They share text, images, videos, audio, and frequently add new content on a daily basis. They may also periodically update existing content, e.g. modifying, replacing, or deleting a variety of datasets. Friends share any information with different levels of addition and update frequencies. A variety of datacenters located around the world can be used by the social network provider. Some geographic locations may have several datacenters and some none. These friends expect to be able to receive updated data including news and event feeds, some made up of data in large size like images and videos, within a very short period of time, no matter where their geographic locations are. Data is accessed often by mobile devices in different locations. Constantly slow updates or problems in timely playing quality videos/audios, viewing images or interacting with other dynamic social media contents are unacceptable to the users.
Furthermore, social networks have a dynamic and growing nature where different scenarios happen. These scenarios are identified as follows in a decreasing order based on their frequency of happening:
**S1:** New datasets are added and existing datasets are updated or deleted
**S2:** Replicas become unsynchronized from time to time and synchronization is required to fulfill the consistency
**S3:** New users join the social network
**S4:** Users create new friendships
**S5:** Workload and access frequencies change over time
**S6:** Existing friendships can be broken
**S7:** Users move and change their locations
**S8:** Existing users may delete their accounts permanently
**S9:** New datacenters might be added or existing datacenters might be removed
Scenarios 1-4 happen more frequently in a social network while scenarios 5-7 happen less frequently. Scenario 8 is a relatively rare scenario and finally, scenario 9 is a very rare scenario yet possible and needs to be considered. These scenarios are explained in more detail with some real world examples in Part 1 of our online supplementary materials. With the emergence of cloud computing [14], social network providers can store data in cloud datacenters at a lower cost. When using geo-distributed cloud datacenters to store social media data, the service provider needs to have the minimum possible number of replicas stored for every user that are capable of ensuring the latency requirement for their friends who are accessing their data. The problem that service providers face is to have the most affordable system by considering the trade-off between monetary cost and latency. **Therefore, we need a minimum cost storage strategy for data placement in the cloud to find the minimum possible number of replicas for every user's data and their locations that can guarantee key service level agreement constraints such as latency and availability**. As social networks are dynamic, replicas and their placement need to be updated based on ongoing changes in the social network over time. Thus, we must make online, real-time replica placement updates.
In this research, we pursue two main objectives. **First**, the bottom line is to keep $P^{\text{th}}$ percentile of individual access latencies lower than 250 ms. With such a short latency, a user will not notice any delay for the best user experience possible, based on research at Google [10]. Considering the individual latency of all friends for all users makes our work much more practical and significantly distinct from others using such as average latency [15]. **Second**, based on achieving the above, the primary goal is to cut the replication cost to a minimum without sacrificing the quality of service, i.e., the user's minimum latency requirement.
There is a huge difference between average latencies and percentile individual latencies considered in this paper. For example, let us consider Amazon datacenter in Sydney with 70% of users located in London and 30% of users located in Melbourne. London users access the data placed in this datacenter in almost 332 ms while Melbourne users access the same dataset in about 56 ms (see Part 2 of our online supplementary materials). The average latency to access the dataset placed in the Sydney datacenter by all users is 249.2 ms, lower than the target requirement of 250 ms. However, in reality, the latency requirement is not met by the majority of the users, i.e., 70% in this example. In contrast, $P^{\text{th}}$ percentile tells us the value greater than or equal to $P$% of our data. For this example, the latency requirement is fulfilled for only $30^{\text{th}}$ percentile of the users, i.e., 30%, which is very poor but reflects reality. In contrast, for a reasonable service level agreement, $P^{\text{th}}$ percentile is normally much higher, such as 90% or above [16]. Note it is theoretically impossible to guarantee latency requirement for all the users, i.e. 100% of users since network failure, while rare, is unavoidable. There may also be some users located in areas with no nearby datacenter, making it impossible to satisfy the target latency for them.

## 3 Problem Formulation

In this paper, we address the research problem of dynamic data placement in social networks, while minimizing monetary expenses incurred in using resources of geo-distributed clouds and guaranteeing social network users' requirements, i.e., latency. We handle update and adaptation of the replication and placement based on the dynamic environment of social networks. For every user, we need to create an initial latency matrix of all their friends from all datacenters. Then, we update the latency matrix for all users based on all changes in the social network over time. The objective is to find and keep the most cost effective placement of data replicas of each user in different datacenters so that $P^{th}$ percentile of individual requests from all friends of the user has the latency requirement fulfilled over time.

### 3.1 Data Placement Formulation

The set of $n$ users in the social network is denoted as $Users = \{1, 2, \ldots, n\}$ and the set of $m$ different datacenters in the cloud environment is represented as $Datacenters = \{1, 2, \ldots, m\}$. Users have relationships with each other, which are described by a matrix of *relationships* with the rows as users and columns as friends. Connections are denoted as $Connections = \{1, 2, \ldots, c\}$, where every element is assigned to a row in the matrix of *relationship* which refers to a connection between a user and a friend. Sets of *Users*, *Datacenters*, and *Connections* are updated once there is a change in any of these sets, such as when a friendship is created/broken, a user has joined/left, or a datacenter is added/removed. Notations used in this paper are shown in the Appendix.

Our overall data placement problem is divided to two stages: static and dynamic placement, described in detail below. Time is modelled as equal time periods *ts*; a static placement is used initially and the dynamic data placement is applied during different time periods to adapt the current placement based on the changes in the social network.

#### *3.1.1 Static Data Placement*

We formulate the initial static data placement problem as a set cover problem. Latency between users and different datacenters (DCs) is denoted as matrix $L$ of size $n \times m$:

$$\forall i \in Users, \forall j \in Datacenters$$
$$L_{ij} = Delay\ of\ user\ i\ accessing\ DC\ j \quad (1)$$

For every user $i$, we find the number of friends $FriendsNum_i$ and let $L'$ present the latency of all his/her friends to access all datacenters which is a matrix of $FriendsNum_i \times m$.

$$\forall i \in Users, \forall j \in Datacenters, \forall k \in FriendsNum_i$$
$$L'_{ijk} = Delay\ of\ friend\ k\ of\ user\ i\ accessing\ DC\ j \quad (2)$$

Finally, for every user, we create a delay matrix $D$ of size $FriendsNum_i \times m$ for all friends of this user. For a given delay requirement, e.g. $P^{th}$ percentile no more than *Delay*, all elements in the delay matrix are compared with $(P/100) \times Delay$. $\forall i \in Users, \forall j \in Datacentres, \forall k \in FriendsNum_i$

$$D_{ijk} = \begin{cases} 1 & if\ L'_{ijk}\ is\ less\ than\ (\frac{P}{100}) \times Delay \\ 0 & Otherwise \end{cases} \quad (3)$$

For every user $i$, we have a set of elements $U_i = \{U_{i1}, \ldots, U_{iFriendsNumi}\}$, i.e., a list of the friends for user $i$ and a set of the subsets of $U_i$ as $S_i = \{S_{i1}, ..., S_{im}\}$, i.e., to store the data of user $i$ in any of datacenters $1$ to $m$. Every element in $S_i$ is considered as a subset of the elements in $U_i$ because by replicating the data of user $i$ in every datacenter in $S_i$, a subset of the friends can have the latency requirement fulfilled. The goal is to find a subset of $S_i$ for user $i$ by which all friends of this user can access his/her data for $P^{th}$ percentile of their requests with latencies no more than the acceptable latency. Our strategy is to find the minimum number of columns, i.e., datacenters in delay matrix $D_{ijk}$ that have the elements with value of 1 covering all rows, i.e., friends. This equates to a "Set Cover Problem" which is NP-Complete [17].

In the set cover problem, we are given a universe $U_i$ for every user $i$, i.e., list of the friends for every user in our problem, such that $|U_i| = FriendsNum_i$, i.e., number of all friends for user $i$, and sets $S_1, \ldots, S_j \subseteq U$, i.e., placement of replicas in all different datacenters which guarantees the latency requirement for a subset of the friends. A set cover is a collection $S$, i.e., the solution set in our problem and includes some of the sets from $S_1, \ldots, S_j$ whose union covers the entire universe $U$. Formally, $S$ is a set cover if $\bigcup_{S_i \in S} S_i = U$. We would like to minimise $|S|$. In order to minimize $|S|$, a weight is defined, i.e., the storage cost of one replica per request in our problem. Therefore, the solution space is as follows:

$$\forall i \in Users, \forall j \in Datacenters$$

$$S_{ij} = \begin{cases} 1 & Data\ of\ user\ i\ is\ stored\ in\ datacenter\ j \\ 0 & Otherwise \end{cases} \quad (4)$$

$$\sum_{j=1}^{m} S_{ij} \geq MinReplica_i \qquad \forall i \in Users \quad (5)$$

*MinReplica* is set to 2 in order to ensure the availability of data for all users. In a cloud environment, maintaining high data availability is important [18]. Different cloud providers might have their own availability policy by keeping a certain number of replicas for data. A minimum of 2 replicas are considered in our simulations. Therefore, for every user *i*, if the final optimal number of replicas $ReplicaNum_i$ is less than $MinReplica_i$, ($MinReplica_i$-$ReplicaNum_i$), i.e., 1, an extra replica will be placed in user's nearest datacenter not currently holding any replica.
Our static strategy can find the most affordable data placement strategy and guarantee the latency requirement for static social networks. However, it is not practical in dynamic social networks and therefore is used only to generate the initial placement for our dynamic data placement and replication strategy.

*3.1.2 Dynamic Data Placement*
After generating the initial placement of replicas for the social network using our static strategy, the next step is to adapt the replica placement based on changes to the social network in an ongoing fashion so that we can have our key latency requirement fulfilled with minimum cost over time. Our problem is a fully dynamic set cover problem [19] in which for every user, not only the subset of the universe, *S*, changes but also the universe itself, *U*, changes over time. In our domain, this dynamic set cover strategy is applied to different users, who also change over time, and users are recognized by location, which is dynamic as well.
Requests from different friends need to be routed to suitable replicas. We create a replica access table for every user in which requests are accessed from the nearest datacenter holding any replica of the requested data. At the end of different time periods, by having the final number of replicas, their locations, and the replica access table, the final latency and cost which are modelled in Sections 3.2.1 and 3.3.1 can be calculated. Efficiency and effectiveness objectives are modelled in Sections 3.2 and 3.3 respectively.

3.2 Efficiency
Latency and time overhead are defined here, utilized in Section 4 and evaluated in Section 5. Latency is the time for users to access data. Time overhead is (1) the overall time for our static strategy to find the best solution and (2) the update time taken for our dynamic strategy to adapt the solution.

*3.2.1 Latency*
In this work, latency between users and datacenters is determined by using the actual latency of real end users in different locations around the world to access different cloud datacenters. Given a placement of data in different datacenters, every user accesses data from the nearest datacenter that holds a replica of the data. Thus, the final latency for every user is $P^{th}$ percentile of latencies of all requests from all friends to access this user's data. As the percentage of more than 90% makes much more sense in most applications [16], requirements are set as 90% and 99.9% (also 95% and 99%) of the latencies of no more than 250 ms. The goal is to have $P^{th}$ percentile of individual latencies (for all users and all their friends) of no more than the acceptable delay:

$$P^{th}(latency_r(t)) \leq Delay \quad (6)$$
where $r = 1, \ldots, \sum_{i=1}^{n} \sum_{j=1}^{FriendsNum_i} RequestNum_{ij}\ (t)$

The latency of request *r* in time period *ts* is the time for the friend sending a request to access the data from the nearest datacenter containing the replica of the requested data. Different users have friends with different access frequencies, i.e., the number of times they access this user's data. Access frequencies of friend *j* of user *i* in time period *ts* is shown by $RequestNum_{ij}(ts)$. The total latency for every user affects more by the friends who access this user's data more frequently.

*3.2.2 Time Overhead*
On one hand, we need to ensure the latency requirement for users. On the other hand, we should not jeopardize the time it takes for our strategy to solve the problem. The amount of time taken to find the initial solution $S_0$ and update the solution $S_{t-1}$ to $S_t$ is measured in order to model the time overhead. We use greedy and dynamic greedy algorithms with polynomial time approximation, as evaluated in Section 5.

3.3 Effectiveness

To ensure the effectiveness of the final solution, cost, competitive ratio, and recourse are defined here, utilized in Section 4 and evaluated in Section 5. Cost is the rate for storing, requesting, transferring data replicas from different datacenters, and finally synchronizing different replicas. Competitive ratio is the worst-case ratio between the cost of our strategy and the optimal strategy. Recourse is the number of replicas added or dropped from the solution.

*3.3.1 Cost*

With $n$ users, the cost rate in time period $ts$ is the total monetary cost of storing replicas of all users' data in different datacenters during this time period, requesting and transferring all users' and friends' data replicas from different datacenters, and synchronizing different replicas from the primary replica for all the users. The total cost is calculated as follows:

$$TotalCost(\$) = \int_{ts=1}^{T} \sum_{i=1}^{n} (StorageCost_i(ts) + TransferCost_i(ts) + UpdateCost_i(ts)) \times dt \quad (7)$$

where firstly, the storage cost is:

$$StorageCost_i(ts) = \sum_{j=1}^{m} S_{ij} \times (UnitWRequestPrice_j(ts) + UnitStoragePrice_j(ts) \times StoredDataSize_i(ts) \quad (8)$$

$UnitWRequestPrice_j(ts)$ is the price in time period $ts$ for write request in datacenter $j$. $UnitStoragePrice_j(ts)$ is the price for storing one GB of data at the end of time period $ts$ in datacenter $j$. $StoredDataSize_i(ts)$ is the data size for user $i$ at the end of time period $ts$. Therefore, the storage cost is the cost for requesting to write and store a user's data and replicas in different datacenters during a time period.

Secondly, the transfer cost is calculated as follows:

$$TransferCost_i(ts) = \sum_{k=1}^{FriendsNum_i} \left( RequestNum_{ik}(ts) \times \left( UnitRRequestPrice_{RT_{ik}}(ts) + UnitTransferPrice_{RT_{ik}}(ts) \times StoredDataSize_i(ts) \right) \right) \quad (9)$$

$UnitRRequestPrice_j(ts)$ is the price in time period $ts$ for read request in datacenter $j$. $UnitTransferPrice_j(ts)$ is the price for transferring one GB of data from datacenter $j$ in time period $ts$. Moreover, $RT$ is the replica access table and $RT_{ik}$ shows the datacenter from where friend $k$ reads the data of user $i$. Thus, the transfer cost is the cost for requesting and transferring a users' data and replicas from different datacenters during one time period.

Finally, the update cost is calculated as follows:

$$UpdateCost_i(ts) = UnitRRequestPrice_{DC_{main_i}}(ts) + UnitTransferPrice_{DC_{main_i}}(ts) \times StoredDataSize_i(ts) + \sum_{j=1, j \neq DC_{main_i}}^{m} S_{ij} \times (UnitWRequestPrice_j(ts) + UnitStoragePrice_j(ts) \times (StoredDataSize_i(ts) - StoredDataSize_i(ts-1)) \quad (10)$$

$DC_{main_i}$ is the datacenter where the primary replica of user $i$ is located, i.e., main datacenter for user $i$. $StoredDataSize_i(ts\text{-}1)$ is the data size for user $i$ at the end of time period $ts\text{-}1$ or in the beginning of time period $ts$. Consequently, the update cost is the cost for synchronizing users' replicas from their primary replicas at the end of a time period. More specifically, the synchronization cost includes the cost for requesting and transferring users' primary replicas from their main datacenters and requesting to write and store user's new data in different datacenters. The total cost is the summation of the storage cost, transfer cost, and update cost during different time periods. For the initial static data placement, as there is no synchronization required, the update cost is not applicable.

*3.3.2 Competitive Ratio*

Online algorithms are studied from the viewpoint of competitive analysis in [20, 21]. The competitive ratio of an online algorithm for an optimization problem is defined as the approximation ratio achieved by the algorithm, that is, the worst-case ratio between the cost of the solution found by the algorithm and the cost of an optimal solution. The greedy algorithm is proved to be an O(log$n$)-approximation algorithm for the set-cover problem [21], thus the solution of this algorithm can be no more than log($n$) times worse than the optimal solution in the worst case. The optimal solution is considered as the log($n$)×(static solution).

*3.3.3 Recourse*

Recourse is the number of sets added or dropped from a set cover over the course of the algorithm, as a function of the length of the input sequence. An online algorithm is $a$-competitive with $r$ recourse if at every time period $ts$, solution $S_t$ has the total cost at most $a.Opt_{ts}$, and the total recourse in the first

batch of *ts* time periods is at most *r.ts*. For *r* worst-case recourse, the number of sets dropped at each time period must be at most *r*. The recourse of our strategy is also evaluated in Section 5.

## 4 Our Dynamic Data Placement Approach

Dynamic set cover is used to optimize the cost of data placement in social network services with interconnected datasets. Static data placement is first used to find the initial data placement. Then, our fully dynamic adaptation strategy is divided to two phases of eager and lazy adaptations. Eager adaptation is done on the fly except for the scenarios of synchronizing the replicas (S2, i.e., scenario 2 introduced in Section 2) and adapting the workload and access frequencies of the friends (S5). For S2 and S5, due to high rates of change and lower frequency, the adaptation is postponed and done during different time periods, so called lazy adaptation. Lazy adaptation is based on the greedy algorithm while eager adaptation is based on either greedy or dynamic greedy algorithms. Fig. 1 provides an overview of our strategy.

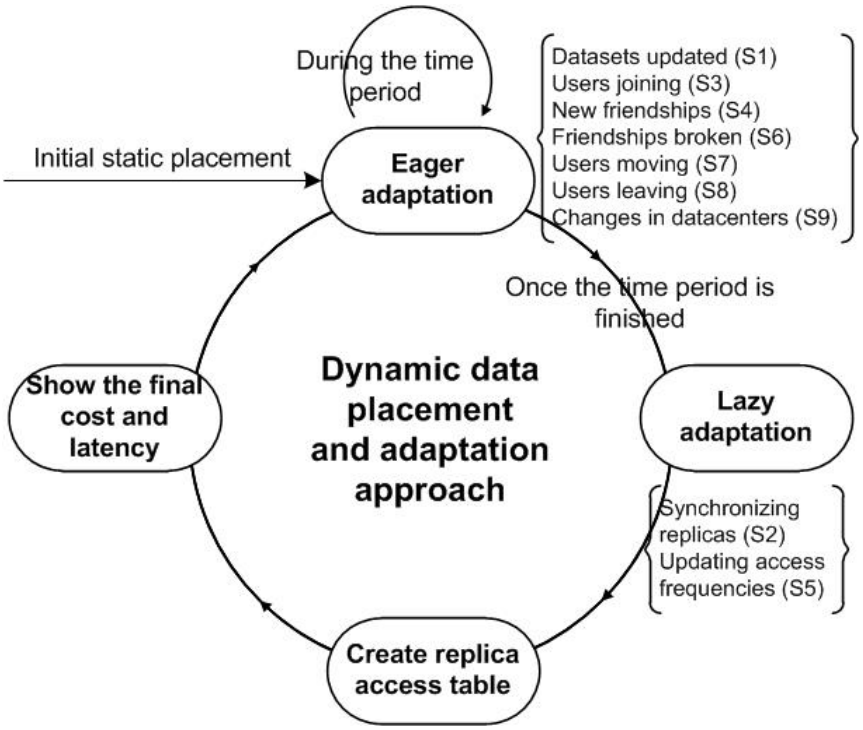


Fig. 1. Dynamic data placement process

4.1 Initial Static Data Placement

The concept of set cover is used to optimize the initial cost of data placement. A greedy algorithm is used to solve the initial static social network data placement problem. This algorithm iteratively picks the most cost-effective set that contains the largest number of uncovered elements in each stage, and removes the covered elements, until all elements are covered. Let $I$ be the set of elements already covered at the beginning of an iteration. Cost effectiveness of set $S$ is the average cost at which it covers new elements, i.e., $Cost(S)/|S-I|$. The weight of an element is the average cost at which it is covered. Equivalently, when set $S$ is picked, we can think of its cost being distributed equally among the new elements covered, to set their weights [11].

For every user $i$ we find set $S_i$, i.e., the placement of user $i$'s data in different datacenters. Weight of every solution in the greedy algorithm is considered as the storage cost of replicas in different datacenters divided by the number of requests being accessed within the acceptable latency.

4.2 Overview of Dynamic Greedy Algorithm

An obstacle to make greedy algorithms dynamic is their sequential nature and insertions/deletions of elements can further disorganize the sequence. However, greedy algorithms can be maintained fast, and with small amounts of recourse using simple "local" moves [19].

In the dynamic greedy algorithm presented in [19], the input is a set system $(U, S)$, which is the solution of a static set cover. The input sequence is $\sigma = <\sigma_1, \sigma_2, \ldots>$, where request $\sigma_t$ is either $(e_t, +)$ or $(e_t, -)$. $A_t \subseteq U$ denotes the active elements at time $t$ with the initial active set as $A_0 = \emptyset$. If $\sigma_t = (e_t, +)$, then $A_t \leftarrow A_{t-1} \cup \{e_t\}$; if $\sigma_t = (e_t, -)$ then $A_t \leftarrow A_{t-1} \setminus \{e_t\}$. At time $t$, only the elements seen so far and which sets they belong to are known, and there is no need to know either $U$ or $S$ upfront. When a new element arrives, it reveals

the sets containing it. We maintain a feasible set cover $S_t \subseteq S$, i.e., the sets in $S_t$ must cover the active elements $A_t$.
Let $Opt_t$ be the cost of the optimal set cover for the set system ($A_t$, $S$). Let $n_t$ denote the number of elements that needs to be covered at time $t$, i.e., $n_t := |A_t|$, and $n$ denote the maximum value of $n_t$, i.e., $n = \max_t n_t$. For the fully dynamic set cover problem, there is an O(log $n$) competitive deterministic algorithm for the dynamic set cover problem, with O(1) non-amortized recourse per input step. If all sets have the same cost (unweighted set cover), then the competitive ratio improves to O(1). Our greedy algorithm chooses sets one by one, minimizing the incremental cost-per-element covered at each step. Analysis shows that the number of elements covered at incremental-costs $\approx 2^i(Opt/n)$ is at most $n/2^i$, which will give the desired O(log $n$) bound for approximation factor. The key steps of this algorithm are described in [19].

4.3 Our Dynamic Data Placement Approach
After finding the initial data placement using our static data placement strategy, equal time periods are considered. For every user $i$ we find set $S_i$, i.e., the placement of user $i$'s data in different datacenters. Users, connections, datasets, and the datacenters are updated once a change happens in the social network. To find a suitable solution based on different scenarios, the dynamic strategy is divided to two phases of eager and lazy adaptations.
Eager adaptation is iteratively applied during a time period in order to update the solution based on any changes in datasets (S1, i.e., scenario 1 introduced in Section 2), users joining (S3), new friendships (S4), friendships broken (S6), users moving (S7), users leaving (S8), and changes in datacenters (S9). Once a time period is finished, lazy adaptation is applied in which synchronization is done (S2) and the solutions are updated based on the new workload and access frequencies (S5). Finally, the replica access table is updated and the final cost and latency for the current placement is calculated and shown.

*4.3.1 Eager Adaptation*
Eager adaptation, as depicted in Fig. 2, is used to address social network changes for scenarios 1, 3, 4, 6-9 from Section 2.1. The replica placement adaptation is done on the fly during different time periods. For all the scenarios, the list of the users, datasets, connections, and datacenters are updated when needed. Then the suitable action is taken based on the scenario. The replication strategy updates are explained below for each corresponding scenario:
**S1.** Adding/Deleting datasets: Data sizes increase on a daily basis, i.e., at any time during different time periods, for all the users based on their workload. Data sizes and storage costs are updated accordingly.
**S3.** Joining of users: Once a new user joins, latencies for accessing different datacenters, list of friends for this user, initial data storage size, matrix of friends' delays, workload and access frequencies of the friends are created. The data placement problem is mapped to a set cover problem for the user and the greedy algorithm is used to solve it.
**S4.** New friendships: Once a new friendship is created, a list of friends and access frequencies for the user who created the relationship and the replica access table is updated to find the nearest datacenter to this new friend. If the latency requirement is fulfilled for this user with the current solution, no new replica is created. Otherwise, the problem is mapped to a dynamic set cover problem and dynamic greedy algorithm is used to solve it.
**S6.** Breaking friendships: When users unfriend each other, the problem is mapped to a dynamic set cover problem using the dynamic greedy algorithm to solve it.
**S7.** Changing user's location: Users may move temporarily or permanently. Once a user moves, a replica is created in the nearest datacenter as the new primary replica and all write requests for this user directed to the new primary replica. The replica access table is updated and the user is asked if the change is temporary or permanent. For a permanent move, and there is no request from friends for the

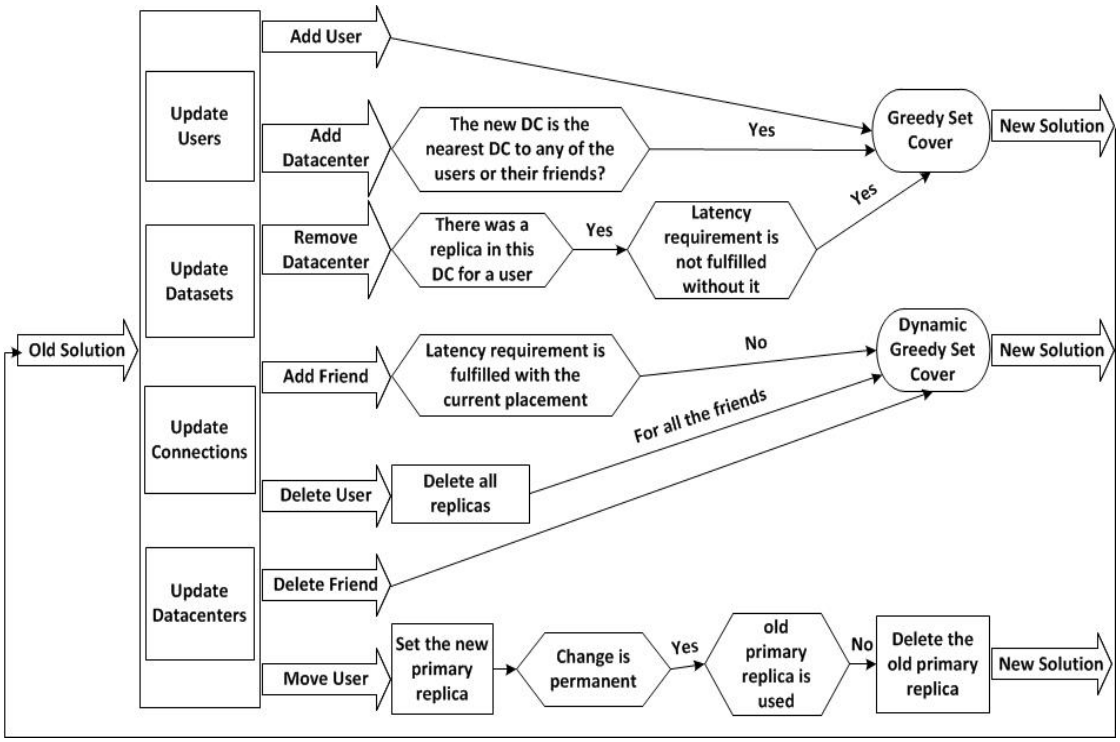


Fig. 2. Eager adaptation process

old primary replica, the old primary replica is deleted.

**S8.** Leaving of users: Quitting could be temporary or permanently deleting the account. Temporary quits, e.g. deactivating in Facebook, are ignored and data are kept in case they re-join. For permanent quits, we need to delete all the replicas for the user but also the "breaking friendship" scenario needs to be considered for all the friends of this user.

**S9.** Adding/Removing datacenters: If a new datacenter is added, the sorted list of datacenters for all the users needs to be updated. If the new datacenter could be a primary datacenter for any users or their friends, the placement is redone for this user using the greedy algorithm. If one of the current datacenters is removed or unavailable, the replica access table is updated to redirect the requests of this datacenter to another datacenter. If there is a user who cannot have the latency requirement fulfilled due to the removed datacenter, a new solution needs to be found for this user using the greedy algorithm.

*4.3.2 Lazy Adaptation*

We use lazy adaptation, as depicted in Fig. 3, to address scenarios 2 and 5 from Section 2.1. The adaptation is done at the end of every time period and adaptation is only applied to the necessary users without any static replication with complete re-computation needed at any point:

**S2.** Synchronization of replicas: During every time period, all the write requests from the users are routed to the primary replicas and the read requests from the friends are redirected to either the primary replica or one of the secondary replicas based on their proximity. At the end of every time period, the

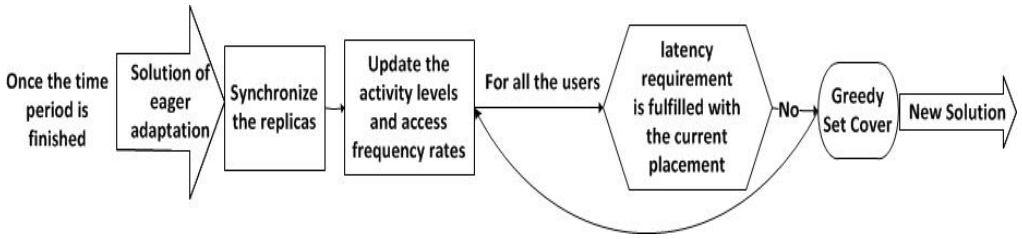


Fig. 3. Lazy adaptation process

secondary replicas of all users are synchronized with the primary replicas and update cost is calculated. Synchronization could be done in an eager fashion as there could be some inconsistency during that specific time period. However, it is not very critical in social network applications since eventual consistency suffices [22].

**S5.** Changes in workload and access frequencies: Since the workload and access frequencies change for all the users and their friends, it is not possible to adapt the solution on the fly based on such changes. Therefore, at the end of every time period, the solution is adapted based on the changes in workload and access frequencies of friends. Instead of adapting the solution for all users, the replica access table is updated for the users given the new workload, access frequencies and existing solution. If the latency requirement is not fulfilled for any users with the current solution, the greedy algorithm is used to find a solution for the user using the new workload and access frequencies.

*4.3.3 Data Placement Strategy*

Our strategy is described by the pseudocode in Algorithm 1 whilst details of our static data placement, eager adaptation, and lazy adaptation algorithm pseudocodes are presented as Algorithms 2, 3 and 4 in Part 3 of online supplementary materials.

ALGORITHM 1

DYNAMIC DATA PLACEMENT STRATEGY PSEUDOCODE

**Inputs:**
Social network graph (*Users* and *Connections*)
Existing solution set {$S_1$, $S_2$, ..., $S_m$} for every user
Time period duration: *interval*
Number of connections: *c*, users: *n*, and datacenters: *m*
Latency requirement with *Delay* and $P^{th}$ percentile
**Outputs:**
Adapted solution set {$S_1$, $S_2$, ..., $S_m$} for every user
Replica access table
Cost and latency of the final placement

**Algorithm**

```
1. StaticDataPlacement();        //Referring to Algorithm 2
// Check for the changes until time period is finished
2. do
3.   StartTime = the current time;
4.   while ((CurrentTime - StartTime) <= interval)
5.      EagerAdaptation();       //Referring to Algorithm 3
6.      CurrentTime = the current time;
7.   end while
8.   LazyAdaptation();           //Referring to Algorithm 4
// Update the replica access table
9.  Let RT represent the replica access table
10.  for all connections c'=1 to c of user i and friend j
11.     RTc = the datacenter with the lowest latency for user j which holds a replica of user i
12.  end for
13.  Update and return the final cost and latency
14.  Return the solution set for every user
15.  Return the number of replicas for every user
16.  Return replica access table
17. end do
```

Algorithm 1 is explained below:

1) The initial social network graph of users (*Users*) and their connections (*Connections*), the current number of connections (*c*), users (*n*), datacenters (*m*), existing solution set $S_i$ for every user *i*, in addition to the duration of time periods (*interval*) are retrieved as inputs.
2) Static data placement is initially carried out (line 1)
3) The following instructions are repeated (lines 2-17)
   a) Start of the time period (*ts*) is recorded and while *ts* is not finished, i.e., the duration of *ts* is lower than *interval*, repeat the following instructions (lines 3-7)
      - Call *EagerAdaptation*() to adapt the solution eagerly
   b) Call *LazyAdaptation*() to adapt the solution at the end of time period (line 8)

c) Using the final replicas, the replica access table is updated (lines 9-12). To create the replica access table, the connections are checked one by one and the datacenters are sorted for every connection. The nearest datacenter for every friend holding any replica of every user is used to access the data.
d) The final cost and latency of all the users are updated in this step. Finally, the solution with the number of replicas, cost, latency and the replica access table for the current time period are returned (lines 13-16).
e) Continue to the next time period (line 17)

*4.3.4 Time Complexity Analysis of our Strategy*

Time complexity of our dynamic strategy, explained in Algorithm 1, is analyzed here. Please refer to Part 3 of online supplementary materials for the time complexity analysis of our static data placement strategy. As discussed earlier, the greedy algorithm has been shown and proven to be an O(log($n$))-approximation algorithm for solving the set-cover problem [21]. Moreover, the results presented in [19] which are based on a novel dynamic greedy framework for the set cover problem, obtains O(log($n$))-competitive results, which leads to a time complexity of O(log($F$)) for dynamic greedy algorithm for every user. Given the time complexity for greedy algorithm in our problem is O($n$×log($F$)) where $n$ is the number of users, $m$ is the number of datacenters and $F$ = max(*FriendsNum*), the overall time complexity for the initial data placement is effectively O($n$×$F$+($c$+$n$)×$m$×log($m$)). The time complexity of our dynamic data placement strategy is proved to be O(((($c$×$m$+$n$)×log($m$)+$n$×log($F$)) as follows.
Time complexity of our dynamic data placement strategy is the time complexity of Algorithm 1. For doing so, we need to find out the time complexity of Algorithms 3 and 4 first. The overall time complexity for Algorithms 3 and 4 are proved as O(($c$×$m$+$n$)×log($m$)+$n$×log($F$)) and O($n$×log($F$)+$c$×$m$×log($m$)) in Sections 3.4 and 3.6 of online supplementary materials.
Therefore, the time complexity of Algorithm 1 is the time complexity of Algorithms 3 and 4 in addition to update the replica access table with O($c$×$m$×log($m$)) and to calculate the total cost and latency with O($n$) that is O(((($c$×$m$+$n$)×log($m$)+$n$×log($F$)). Finally, the overall time complexity of our dynamic data placement strategy considering $\alpha$ as the number of changes in the network, is O($\alpha$×((($c$×$m$+$n$)×log($m$)+$n$×log($F$))).

## 5 Simulation Evaluation and Results

We describe detailed simulation results using real world datasets and using different percentiles of latency requirement. We compare our static strategy with other representative counterparts while comparing the performance of our dynamic strategy with our static strategy under a set of different dynamic scenarios in the social network. Two real world social network datasets are used to demonstrate how our dynamic strategy finds an efficient and effective placement of data with minimized cost while satisfying the latency requirement. The first dataset used is a Facebook social network graph [12] with 63,731 nodes, i.e., users, and 1,545,686 edges, i.e., connections. The second dataset used is SNAP location based Gowala social network graph [13] with 196,591 nodes and 950,327 edges.

5.1 Experimental Settings

In the experiment, we simulate a social network over one year as close to reality as possible by using real world data. We have considered 365 time periods, denoted as timeslots in the experiments, each takes about 100 seconds and is mapped to one physical day in order to simulate a real social network over one year. In every timeslot, all scenarios introduced in Section 4.3.1 could possibly happen at any time during the timeslot and the solution is adapted on the fly once these changes occur. At the end of every timeslot, the solution is adapted based on the scenarios introduced in Section 4.3.2. The growth rate of the users is considered as **18 percent increase year over year,** based on a Facebook report published on May 2017 [23]. We consider 28 percent of the friends for different users as the initial number of the friends based on a report indicating 28 percent of the friends for every user to be "genuine", or close friends [24]. The number of friends is randomly increased over time because the growth rate of the friends depends on factors such as days they were active, contents uploaded, and so on [25]. We assume dynamic scenarios randomly happen during different timeslots based on the frequency of nine scenarios (S1 to S9), as in Section 2.1. Addition and removal of datacenters are considered to happen only once during our simulations.
For simulation, real Amazon datacenters in Virginia, California, Oregon, Ireland, Frankfurt, Singapore, Sydney, Tokyo, and Sao Paulo are considered and the real unit storage cost for data storage per GB per month, request cost per request and transfer cost per GB in all these datacenters are taken into account. The prices of $UnitStoragePrice_j$, $UnitRequestPrice_j$, and $UnitTransferPrice_j$ for Amazon datacenters used in our experiments are shown in Part 4 of online supplementary materials. The final cost of our strategy is the summation of the storage cost; transfer cost; and update cost. To find the latency of these

datacenters, we had 19 users in 19 cities around the world pinging different Amazon datacenters from their locations for ten times. As mentioned before, the average latencies are shown in Part 2 of online supplementary materials. To assign these latencies to a different number of users in these locations, we used different normal distributions with the collected latencies as the mean for every region. Number and location of the users as well as different parameters such as delay time and processing time can vary in our model.

TABLE 1
PERCENTAGES OF USERS WITH DAILY ACTIVENESS LEVELS

| Percentage | Activeness level |
|---|---|
| 8 | 12 |
| 15 | 8 |
| 12 | 4 |
| 14 | 2 |
| 27 | 1 |
| remaining | 0 (inactive, hence not in simulation) |

To measure the workload of the social network, we define a term called activeness level. Activeness level of the users is based on how many times they check their accounts per day. Percentages of different Facebook users and the number of times they check their accounts daily is reported in [26]. We use the same proportion for different percentages of the users, as in Table 1. To find the access frequencies of different users to access different friends' data, the access frequencies of users to access their friends' data is set randomly between 0 and every user's activeness level. The activeness levels and access frequencies may change at the end of each timeslot.

5.2 Simulation Results with Facebook Dataset

The Max Plank institute Facebook dataset [12] with total 63,731 users and 1,545,686 connections are considered as the final social network graph after finishing the simulation of a social network for one year. Based on the statistics presented in Section 5.1, the initial number of users is considered as 54,005, which is 84.74 of the total number of users by incorporating 18 percent annual growth. The initial number of friends for every user is considered as 28 percent of the total friends of the user. By having 84.74 percent of the users and 28 percent of their friends, our simulations start with 29 percent of the whole dataset and would reach to the total dataset after 365 timeslots.

While the users' location information is not provided in the dataset, we generate random locations in 19 different cities as shown in Part 2 of online supplementary materials based on real distribution of Facebook users' locations [27] which is shown in Part 5 of online supplementary materials. We use a similar proportion of the users' locations consistent with that of Facebook and random locations are added to the list. Users' locations are randomly changed for some of the users during different timeslots. Parameters such as pricing and latency from Amazon cloud datacenters are used. Using statistics for 2016 [9], Facebook generates four new petabytes of data per day for the 1.083 billion daily active users. Hence, on average, every active user stores around 3.6 MB (4 PB/1.083 billion) information daily in Facebook datacenters. We generate random sizes of data for users following a normal distribution with average size as the mean initially. Data size increases daily during different timeslots, for all users based on activeness levels.

*5.2.1 Simulation Results for Our Static Strategy*

We compared our static strategy with 11 different strategies. The minimum number of replicas for all strategies are considered as 2 in order to ensure the data availability. These strategies (A1-A12) are listed as follows:

*Random-based strategies:*

**A1**: Random number of replicas (between 2 and the number of datacenters) in random datacenters.

**A2**: Two replicas in two random datacenters.

**A3**: Three replicas in three random datacenters.

*Full-replication strategy:*

**A4**: Full replication of data in all datacenters that is claimed in [28] and [29] as the data placement strategy used for Facebook. Full replication has the lowest possible latency and can be deemed as the minimum latency benchmark.

*Distance-based strategies:* Datacenters are sorted based on the distance for every user as *list1*. Because long distance incurs high latency, users prefer to have a replica of data in their nearest datacenter.

**A5**: Two replicas in the first and second preferred datacenters in *list1*.

**A6**: Three replicas in the three most preferred datacenters in *list1*.

*Friend-based strategies:* Datacenters are sorted based on both distance as *list1* and number of friends as

*list2* for every user. Users prefer to have replicas not only in their nearest datacenters but also in the nearest datacenters to the most of their friends.
**A7**: One replica in the most preferred datacenter in *list1* and one in the most preferred datacenter in *list2*.
**A8**: One replica in the most preferred datacenter in *list1* and two more replicas in the two most preferred datacenters in *list2*.
*Request-based strategies:* Datacenters are sorted based on both distance as *list1* and number of requests as *list3* for every user. Users prefer to have replicas not only in their nearest datacenters but also in the datacenters where most of the requests are from the friends around them.
**A9**: One replica in the most preferred datacenter in *list1* and one in the most preferred datacenter in *list3*.
**A10**: One replica in the most preferred datacenter in *list1* and two more replicas in the two most preferred datacenters in *list3*.
*Social locality based strategy:*
**A11**: *Social locality*, which is maintained in [30], we further consider transfer cost that is ignored in their work.
**A12**: *Our static strategy*.

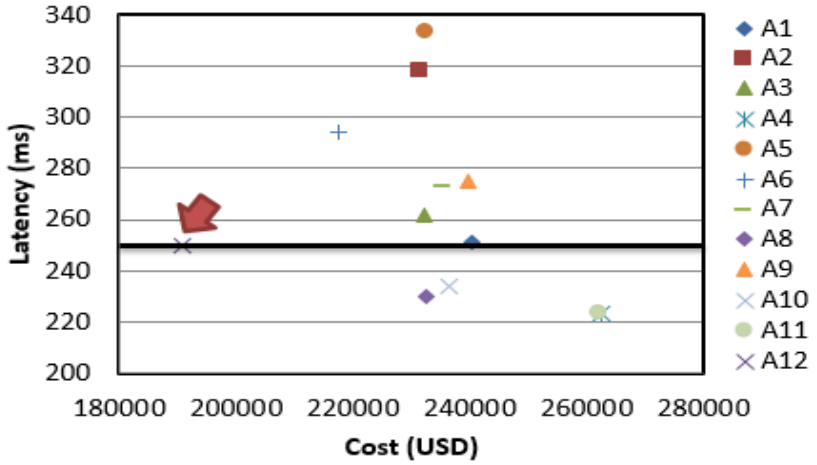


Fig. 4. Latency requirement of 90% lower than 250 ms for Facebook

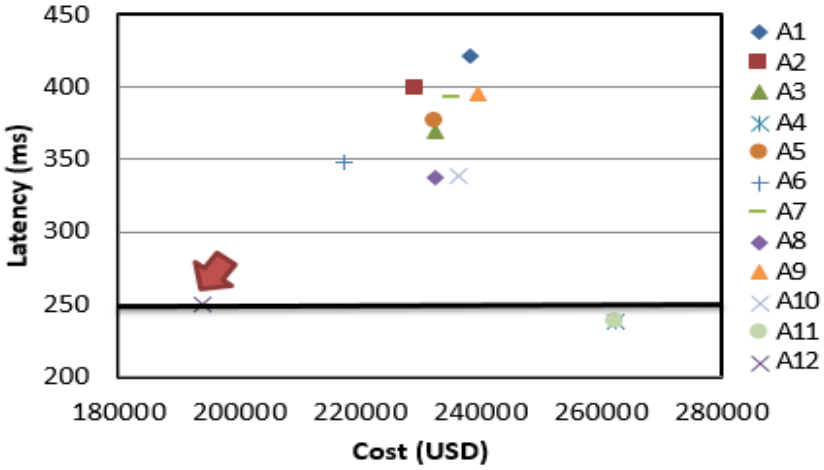


Fig. 5. Latency requirement of 99.9% lower than 250 ms for Facebook

The cost and latency of these strategies, for a duration of one month for the Facebook dataset with different latency requirements are shown in Figs. 4 and 5. An arrow in these figures points to our strategy. As shown in some of the results for the Facebook dataset, having less replicas might cause a higher total cost due to the extra transfer cost since the storage cost is much lower than the transfer cost. For example, strategy A5 with two replicas has a higher cost than strategy A6 with three replicas.
As shown in Figs. 4 and 5, with more results in Part 6.1 of online supplementary materials, our static strategy is able to guarantee the latency requirement in all cases with much lower storage and transfer cost comparing to the other strategies. Moreover, the only strategy, except costly full replication, that can guarantee the acceptable latency for 99.9 percentiles of all requests with a reasonable cost is our greedy set cover based strategy.

*5.2.2 Simulation Results for Our Dynamic Strategy*

The cost and latency for our static, our dynamic, and full replication strategies are compared at the end of different timeslots. Our original static strategy is our reference strategy, which finds the offline solution from scratch based on the existing setting in every timeslot. Our dynamic strategy adapts the solution of the previous timeslot based on all the changes during the current timeslot and the cost and latency of the adapted solution are shown at the end of each timeslot. The full replication strategy, i.e. replication of data in all datacenters, has the lowest possible latency but incurs the highest cost. Several settings based on the latency requirement for users and their friends to access data are used to compare the cost and latency results. Requirements here are set as 90% and 99.9% of individual latencies no more than 250 ms with more results in Part 6 of online supplementary materials.

*5.2.2.1 Simulation results for eager adaptation*

For eager adaptation, the solution is adapted on the fly once any of the scenarios introduced in Section 4.3.1 happens. We simulated four different combinations of the scenarios separately while keeping all other variables fixed for 90 and 99.9 percentiles of latencies. Different experiments are based on a combination of different scenarios, however, S1 and S2 are considered in all experiments. S1, in which datasets are updated, happens with all other scenarios to consider data growth over time. S2 happens at the end of every timeslot to apply replica synchronization. At the end of every timeslot, we update and compare the cost and latency of all three strategies.

The first experiment is to keep datacenters and locations fixed, start from an initial number of connections and add new users and friends until covering all the friends and users. The results are shown in Fig. 6 (Exp 1: Scenarios 1, 2, 3 and 4). The second experiment keeps datacenters and locations fixed, starts from the total number of users and friends and deletes friends and users during different timeslots. Results are shown in Fig. 7 (Exp 2: Scenarios 1, 2, 6 and 8), showing cost minimization while fulfilling the latency requirement when data items change, and users and friends join or leave the network over time.

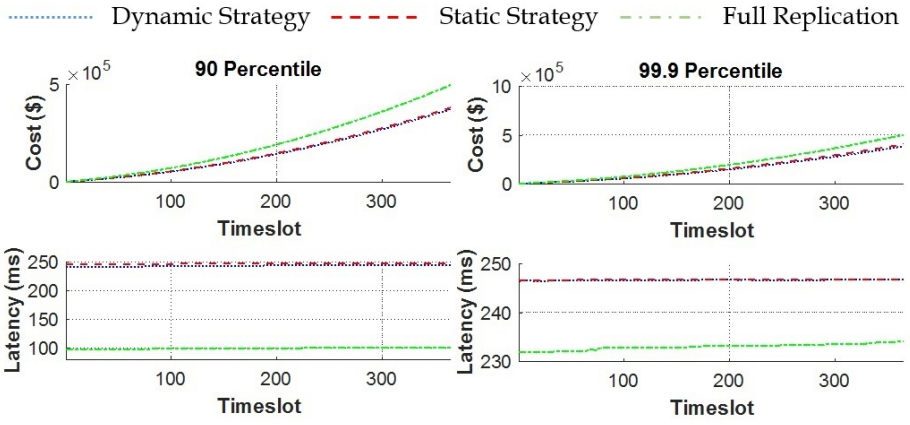


Fig. 6. Cost and latency when new users and friends are added

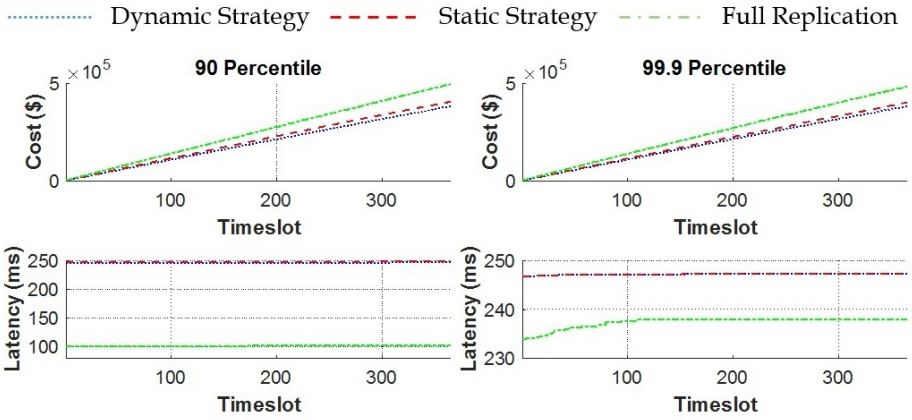


Fig. 7. Cost and latency when users and friends are removed

The third experiment keeps users, friends and datacenters fixed, starts from the total number of users and friends and changes the location of random users during different timeslots. Results are shown in Fig. 8 (Exp 3: Scenarios 1, 2 and 7). This shows cost minimization while fulfilling the latency requirement when users and friends move.

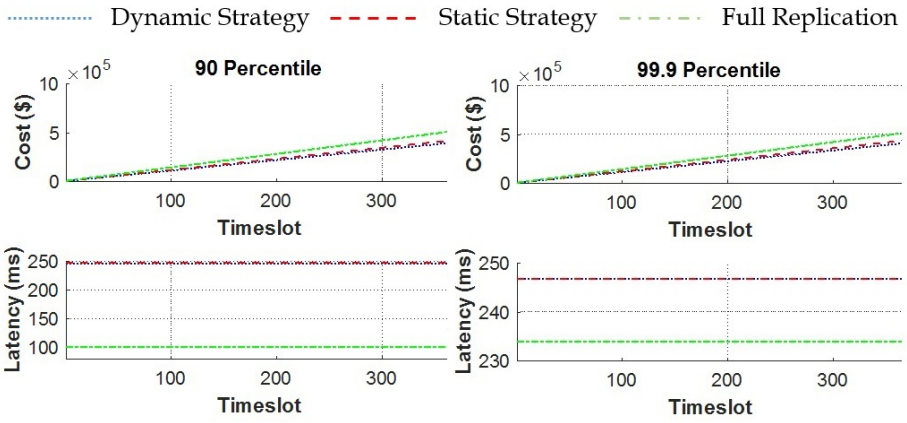

Fig. 8. Cost and latency when users move

The last experiment is to keep the users, friends and locations fixed. Then, we start with the total number of users and friends and (1) the complete list of datacenters, remove datacenters one by one (Exp 4: Scenario 9), and (2) the minimum number of datacenters, add datacenters one by one. (Exp 5: Scenario 10). Results for Exp 4 and Exp 5 are shown in Fig. 9 and Fig. 10 respectively. These show when datacenters are added or removed, our dynamic strategy keeps latency unnoticeable with the minimum cost.

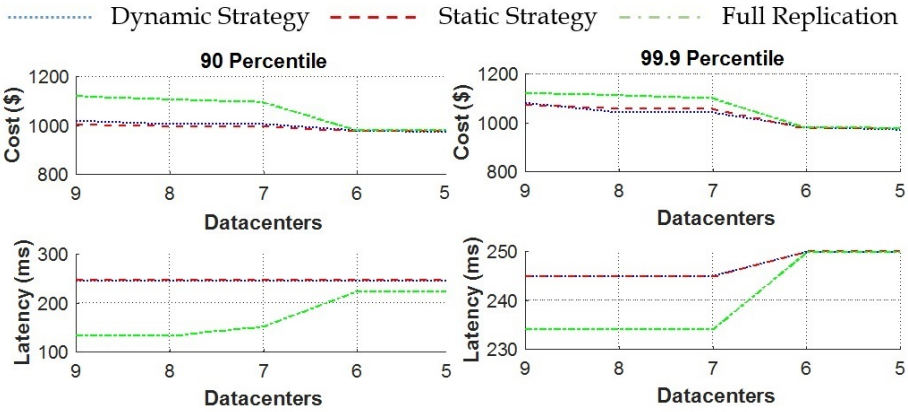

Fig. 9. Cost and latency when a datacenter is removed

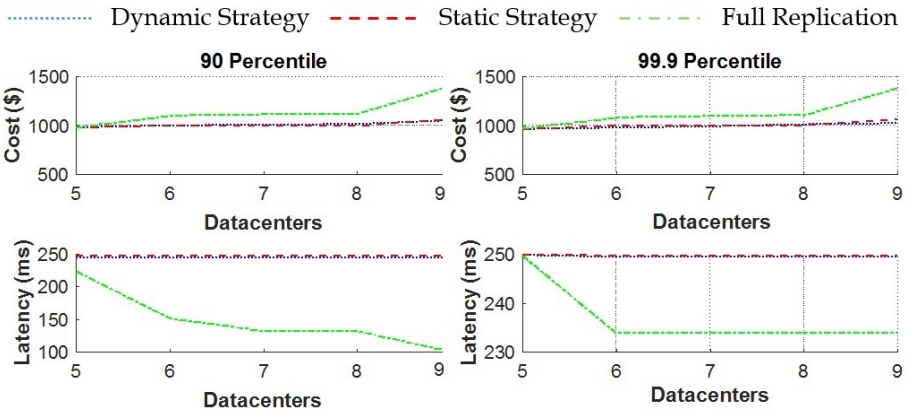

Fig. 10. Cost and latency when a datacenter is added

5.2.2.2 Simulation results for lazy adaptation

For lazy adaptation, the solution is adapted at the end of every timeslot based on the changes in the scenarios introduced in Section 4.3.2. To see how the cost and latency are affected when the workload and access frequencies of different users and friends change, the workload and access frequencies are changed while all other variables are fixed. The cost and latency of the latest solution with the new workload and access frequencies in addition to the cost and latency of static, dynamic, and full replication are shown and compared in Fig. 11 for 90 and 99.9 percentiles of latencies (Exp 6: Scenarios 1, 2 and 5). This shows our strategy fulfills the latency requirement while minimizing the cost when

access frequencies change.

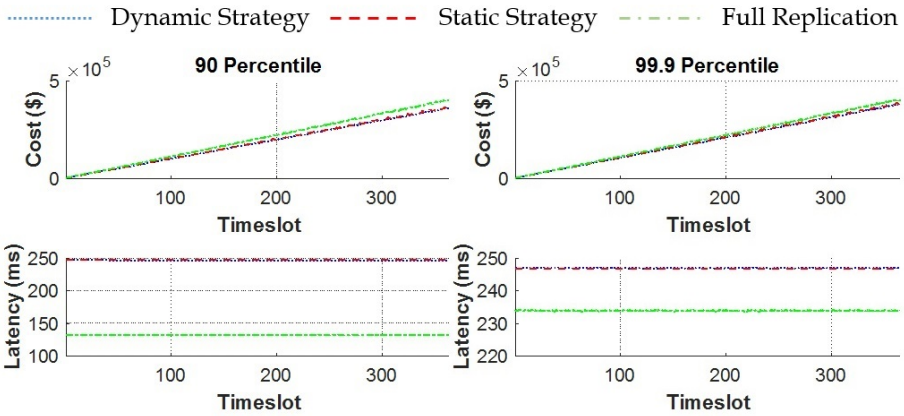


Fig. 11. Cost and latency when access frequencies are updated

*5.2.2.3 Combination of eager and lazy adaptations*

All scenarios are combined (Exp 7: All scenarios) and simulated together with the requirements of 90 and 99.9 percentiles of latencies. The cost and latency of static, dynamic, and full replication, when all scenarios happen together, are shown in Fig. 12. Frequency of the scenarios in different timeslots is shown in Table 2.

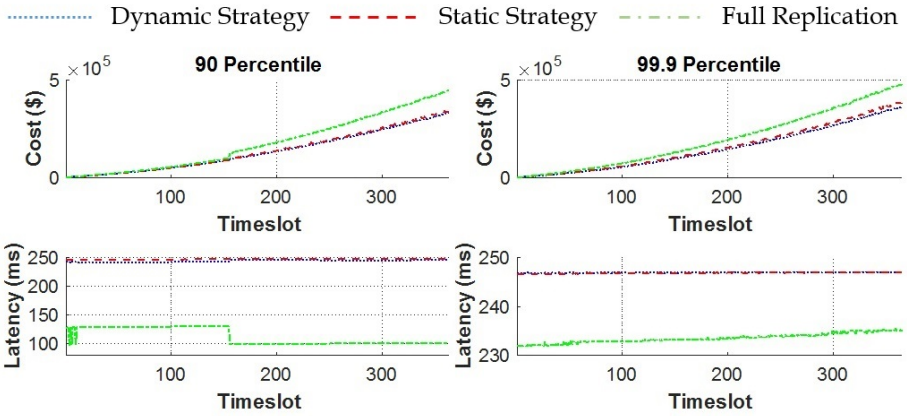


Fig. 12. Cost and latency when all scenarios happen

TABLE 2
FREQUENCY OF DIFFERENT SCENARIOS

| Scenario | Frequency in each timeslot |
|---|---|
| S1 | Average of 3.6 MB of data is added for every user during each timeslot |
| S2 | At the end of each timeslot |
| S3 and S4 | 18%/365 (0.05%) of the initial users and random number of friends are added in each timeslot which is more than 3000 new connections during each timeslot |
| S5 | Random new rates at the end of each timeslot |
| S6 | Random number of friendships between 0-6 in different times during each timeslot |
| S7 | Random number of users between 0-2 in different times during each timeslot |
| S8 | Random number of users between 0-1 in different times during each timeslot |
| S9 | Once randomly in 365 timeslots |

## 5.3 Results Analyses

*5.3.1 Efficiency Evaluation*

Efficiency of our strategy can be evaluated in terms of 1) the time it takes for every user and all their friends to access their data, i.e., achieved latency, and 2) the time it takes to run the algorithm and do the adaptation, i.e., time overhead. It is necessary to not only guarantee the latency requirement for all users by having an optimal data placement, but also to find the optimal data placement in an acceptable time.

From the latency requirement perspective, our strategy is able to guarantee the latency requirement in all cases. The latency requirement is calculated by having *P*, i.e., percentile, and *Delay*, i.e., acceptable latency requirements. As shown in Figs. 4-12 for Facebook dataset and Part 6.2 of online supplementary materials for Gowala dataset, our strategy can guarantee the latency with a much lower total cost.

Moreover, our dynamic strategy is efficient in terms of time overhead. For our simulations, six virtual machines on our local Cloud testbed, all with Intel core i5-4570 CPU, 8 GB RAM Memory, and windows 7 operating system are used. The time it takes for the static strategy to find the initial data placement is about 10 seconds and the time it takes for our dynamic strategy to adapt the solution, as shown in Table 3, for the most frequent scenarios of updating datasets and joining new users/friends is 0.00014 seconds for the Facebook dataset. For the Gowala dataset, it takes about 12 seconds to find the initial data placement using static strategy and 0.00066 seconds to adapt the solution for these most dominant scenarios. Therefore, the results show that it takes only 0.00014 seconds for adapting the solution whilst without having a dynamic strategy to adapt the solution; it takes about 10 seconds by using static strategy to reconstruct the solutions. Thus, our dynamic strategy is about 70000 times more efficient (10s/0.00014s) than the static strategy for Facebook dataset and about 18000 times more efficient (12s/0.00066s) than the static strategy for Gowala dataset.
These results show that although our static strategy is effective and efficient, our dynamic data placement strategy is required to make it practical in dynamic environments with very frequent changes needed. Update time of different scenarios, shown in Table 3, shows that our dynamic strategy is extremely efficient and does not jeopardize the time taken to build the solution in order to have efficient results. Since Exp7 is a mixture of different scenarios and the number and frequency of scenarios are different, it does not make sense to measure and show the update time of Exp7. As results show, our dynamic strategy can save a lot of time compared to using the static data placement strategy.

TABLE 3
UPDATE TIME OF DIFFERENT SCENARIOS

| Update Time (Seconds) | Exp1 (S1, S2, S3, S4) | Exp2 (S1, S2, S6, S8) | Exp3 (S1, S2, S7) | Exp4 (S9.1) | Exp5 (S9.2) | Exp6 (S1, S2, S5) | Exp7 |
|---|---|---|---|---|---|---|---|
| Facebook | 0.00014 | 0.0028 | 0.0124 | 0.21 | 0.55 | 0.08 | N/A |
| Gowala | 0.00066 | 0.0051 | 0.53 | 4.8 | 4.5 | 0.33 | N/A |

*5.3.2 Effectiveness Evaluation*
Effectiveness of our strategy can be evaluated in terms of 1) the total cost of the data placement, 2) the ratio between the cost of our strategy and the optimal strategy, i.e., competitive ratio, and 3) the number of replicas added or dropped from the solution, i.e., recourse.
As shown in Figs. 6-12 our strategy can find the minimal storage, transfer, and update cost while guaranteeing the latency requirement for different percentiles of individual requests based on the latency requirements. Percentages of cost savings compared to the full replication strategy are shown in Table 4. The cost saving percentages for different percentiles of latencies are calculated after finishing the experiments, i.e. 365 timeslots for all the scenarios. Exp4 (adding datacenters) and Exp5 (removing datacenters) happen only once during the experiments.

TABLE 4
COST ANALYSIS OF DIFFERENT SCENARIOS

| Cost Saving | Exp1 | Exp2 | Exp3 | Exp4 | Exp5 | Exp6 | Exp7 |
|---|---|---|---|---|---|---|---|
| Facebook | 26% | 23% | 23% | 26% | 9% | 11% | 26% |
| Gowala | 16% | 20% | 19% | 22% | 10% | 21% | 16% |

The cost ratio between the dynamic strategy and its static counterpart is close to 1 for both Facebook and Gowala datasets in all scenarios. This shows that our dynamic strategy finds a solution as good as the static strategy. As discussed in Section 3.3.2, by having a dynamic to static ratio of 1, the competitive ratio for the dynamic set cover strategy will be $\log(n)$. This means the solution of our dynamic strategy, in the worst case, is $\log(n)$ times worse than the optimal solution.
Finally, recourse, i.e., the number of changes for every user's solution to be adapted in different scenarios is either 0 or 1 which means there is at most 1 change in the replicas in order to adapt the solution. This is a very promising outcome as creating/deleting replicas incurs extra cost, latency and inconsistency.

*5.3.3 Summary*
Based on this analysis, our dynamic strategy is able to find an efficient and effective solution without applying a static data placement and replication from scratch for every social network data change. Therefore, our dynamic strategy is practical for dynamic environments in terms of both efficiency and effectiveness. More detailed results for Facebook and Gowala datasets including the latency requirements of 90, 95, 99, and 99.9 percentiles as well as details of cost analysis are shown in Parts 6 and 7 of online supplementary materials respectively.

Detailed threats to validity for the experiments are discussed in Part 8 of online supplementary materials.

## 6 Related Work

In this section, we compare our work with existing literature in three categories of: social network data placement in cloud, content delivery networks (CDNs), and dynamic data replication in the cloud. In cloud based social network data placement, social locality, i.e. placing the data of all friends in every user's server maintained in [30] and [31] for optimizing multiple social network objectives. However, the social locality assumption incurs a very high cost due to the replication of data in unnecessary datacenters. A multi-cloud hosting system is formulated in [32] and the trade-off between satisfying users' requirements and reducing the inter-cloud data propagation cost is demonstrated. A dynamic, cost-aware, optimized data replication strategy is presented in [18] in which the concept of the knapsack is used to optimize the cost. Additionally, SPAR [33], is based on partitioning of the social network graph as well as replication of users with the aim of ensuring local data semantics, i.e. all relevant data of the direct neighbors of a user are collocated on the same machine as the user. Moreover, [34] suggests that performing partitioning and replication simultaneously can reduce inter-server traffic. However, latency is not considered as a requirement in these papers. The primary focus in [22] is to minimize the cost in latency-sensitive applications as well as consistency and fault-tolerance requirements with considering workload properties. Reducing cloud resources usage, providing good quality of service to users, and minimizing the carbon footprint are considered as a multi-objective optimization problem as addressed in [35]. They consider latency as an objective instead of constraint that makes it not comparable with our work. A geo-cloud based dynamic replica creation in large global Web sites such as Facebook is investigated in [36] and a selective data replication mechanism in distributed datacenters to reduce inter-datacenter communications while achieving low service latency is proposed in [28]. A novel approach of data placement for improving OSN service performance in a Multicloud environment by providing acceptable latency delay is introduced in [37]. The authors claim that since the amount of total data to store is less than the baseline approach, it can also benefit from the storage cost for storing the data. However, minimization of the total monetary cost for replicating data is not considered as an objective in these three papers. To minimize monetary cost, while satisfying user-perceived latency, a geo-partitioning method is employed in [29] which goes beyond the traditional graph partitioning problems. The latency requirement is said to be satisfied as long as each user's data is placed on any one of partitions in a user's vector. Moreover, considering Twitter as their example, it is assumed that users have the same storage cost and therefore the storage cost is simply omitted. Finally, the QoS-aware data replication problem for cost minimization in cloud systems is investigated in [38] but unlike our work for finding the minimum number of replicas for users, the number of replicas is fixed in these papers.

In our earlier work [39], we proposed a Genetic Algorithm (GA) based strategy to minimize the cost while guaranteeing the latency requirement. However, transfer and update costs are ignored and the latency is derived using the distance between user and different datacenters instead of considering real latencies between users and different datacenters. Moreover, based on the scalability problem in using GA, we then proposed a Graph Partitioning (GP) strategy based on users' locations and connections in [40]. A static social network is considered where the users, friendships, and datacenters are fixed and do not change over time. These used a simple data placement and replication strategy, which led us to the current research presented in this paper.

The data replication problem in a CDN or cache network is addressed in many studies [41, 42]. Building CDNs on top of the cloud infrastructure is advocated in [43]. Cache placement is performed in [44] by employing social link information and client preferences. Additionally, a CDN based social video replication and user request dispatching mechanism in the cloud architecture is presented in [45]. Resource provisioning and replica placement problems for cloud-based CDNs are addressed in [46] with an emphasis on handling dynamic demand patterns. Finally, social video distribution over the cloud-centric CDN infrastructure, with the objective of minimizing the overall system monetary cost, while satisfying the averaged time delay is investigated in [45]. Based on [47], CDN mostly handles designing optimal strategies for the case where the number of contents and the scale of user requests are fixed. However, the very challenging issue, which is addressed in this paper is to present a dynamic strategy that can place dynamic contents and requests related to the growing number of users and connections on the fly and continuously ensure the optimality attained by the optimal static solution with complete knowledge of the social network over time. These CDN methods do not consider content dynamicity, which is the most frequent scenario in social networks.

In the category of dynamic data replication in the cloud, an efficient proactive algorithm for dynamic, optimal scaling of a social media application in a geo-distributed cloud is proposed in [47]. Unlike our work, only the number of videos increases in the system, while the total number of users and also the

datacenters are fixed. The problem of social network data placement in a distributed cloud with the aim to minimize the operational cost of a cloud service provider is investigated in [48]. Finally, an associated data placement scheme, which improves the co-location of associated data and the localized data serving while ensuring the balance between nodes is proposed in [49]. Incremental adjustment of replicas is also taken into account in order to deal with workload change. However, unlike our work, changes in the connection links and datacenters are not addressed as part of the dynamic maintenance of the social network. Finally, a lightweight replica placement scheme is proposed in [50] to investigate the latency constraint cost optimization problem in geo-distributed OSNs. However, unlike our work, they do not consider moving of the users and addition/deletion of datacenters. To highlight the importance of datacenter changes, Facebook has recently announced two new datacenter locations, which will cost hundreds of millions of dollars [51]. Even when using cloud datacenters, Amazon as a cloud provider has increased its datacenter locations from 9 in 2014, to 18 in 2018 and is planning to add 4 more datacenters soon [52].
Overall, there is no comprehensive work dedicated to all different scenarios that happen in a social network for data placement, such as moving of the users and addition/deletion of datacenters. Moreover, as mentioned before, we take $P^{th}$ percentile requirement of individual access latencies for all the users into account. Considering all the dynamic scenarios as well as individual access latencies instead of average latencies makes our work novel and much more challenging from other existing works.

## 7 Conclusion and Future Work

In this paper, a dynamic cost-effective data placement problem for social networks is formulated as a dynamic set cover problem, which is NP-complete. Due to the users' dynamic activeness and mobility in social networks, our strategy is applicable in all dynamic environments where users join, leave, move or change their friendships in the social network, and datasets are added, removed and updated as needed. Addition and removal of datacenters are also taken into account. Our novel dynamic data placement strategy is able to adapt according to the changing environment at runtime. A framework consisting a combination of greedy and dynamic greedy algorithms is presented to guarantee that even up to 99.9 percentiles of individual latencies for all requests from different users in the social network is unnoticeable with a minimum cost for storing, transferring, updating, and synchronizing data over time. Compared to alternative full replication, our proposed approach can produce up to 26% cost savings compared to the full replication strategy while meeting latency requirements. Simulation results on two large-scale datasets, Facebook and location based Gowala, with latency timings used from real Amazon cloud datacenters, show the efficiency and effectiveness of our strategy over the duration of one year.
In the future, graph partitioning will be used to shape different groups of users based on their mutual interests, connections, usages, etc. to make our strategy more scalable. Hence, the experiments will be conducted on real social network traces. Using our graph partitioning method, the social network graph will be broken into smaller networks and each network would normally be well within the size range of our experiments. A large social network of users would be manageable by different computers in a parallel and distributed fashion. Additionally, learning methods will be used in order to predict the workload and access frequencies of the friends for future timeslots based on previous timeslots. Moreover, users will be involved in the process of data placement and replication. Thus, users can have different quality of service requirements and expectations. Finally, to improve the efficiency of our strategy in the future, we will apply our method to CDN, e.g. AWS CloudFront [53] or edge/fog computing, e.g. AWS Greengrass [54] in order to bring the advantages and power of the cloud closer to the end users.

**Acknowledgment**

This research is partly supported by the Australian Research Council Linkage Projects scheme under grant LP130100324 and Discovery Projects scheme under grant DP160102412. Grundy and Khalajzadeh are supported by Laureate Fellowship FL190100035.

**References**

[1] S. Kemp. (2017). Digital in 2017, Global Overview. Available: https://www.slideshare.net/wearesocialsg/digital-in-2017-global-overview
[2] J. Constine. (2017). Facebook Now has 2 Billion Monthly Users… and Responsibility. Available: https://techcrunch.com/2017/06/27/facebook-2-billion-users/
[3] W. Li, Y. Yang, and D. Yuan, "Ensuring Cloud Data Reliability with Minimum Replication by Proactive Replica Checking," IEEE Transactions on Computers, vol. 65, pp. 1494-1506, 2016.
[4] D. Yuan, Xiao Liu, and Y. Yang, "Dynamic On-the-Fly Minimum Cost Benchmarking for Storing Generated Scientific

Datasets in the Cloud," IEEE Transactions on Computers, vol. 64, pp. 1-15, 2015.
[5] M. D. Assunção, R. N. Calheiros, S. Bianchi, M. A. S. Netto, and R. Buyya, "Big Data Computing and Clouds: Trends and Future Directions," Journal of Parallel and Distributed Computing, Volumes 79–80, pp. 3-15, 2015.
[6] Amazon S3. Available: http://aws.amazon.com/s3
[7] Google Cloud Storage. Available: http://cloud.google.com/storage
[8] Windows Azure. Available: http://www.microsoft.com/windowsazure
[9] K. Smith. (2016). Marketing: 47 Facebook Statistics for 2016. Available: https://www.brandwatch.com/blog/47-facebook-statistics-2016/
[10] J. D. Brutlag, H. Hutchinson, and M. Stone, "User Preference and Search Engine Latency," in JSM Proceedings, Quality and Productivity Research Section, Alexandria, USA, 2008.
[11] T. Roughgarden, "A Second Course in Algorithms: Linear Programming and Approximation Algorithms," Lecture Notes, Stanford University, 2016. Available: http://theory.stanford.edu/~tim/w16/1/l17.pdf
[12] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi, "On the Evolution of User Interaction in Facebook," in ACM SIGCOMM Workshop on Social Networks (WOSN'09), Barcelona, Spain, 2009, pp. 37-42.
[13] E. Cho, S. A. Myers, and J. Leskovec, "Friendship and Mobility: User Movement in Location-Based Social Networks," in ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), San Diego, USA, 2011, pp. 1082-1090.
[14] A. Weiss, "Computing in the Clouds," netWorker, vol. 11, pp. 16-25, 2007.
[15] Z. Zeng and B. Veeravalli, "Optimal Metadata Replications and Request Balancing Strategy on Cloud Data Centers," Journal of Parallel and Distributed Computing, Volumes 74, pp. 2934–2940, 2014.
[16] X. Liu, J. Chen, and Y. Yang, "A Probabilistic Strategy for Setting Temporal Constraints in Scientific Workflows," in Business Process Management. vol. 5240, ed: Springer Berlin Heidelberg, 2008, pp. 180-195.
[17] S. Chawla, (2007). Advanced Algorithms: Greedy Algorithms, Divide and Conquer, and DP [Lecture notes]. Retrieved from http://pages.cs.wisc.edu/~shuchi/courses/787-F07/scribe-notes/lecture02.pdf
[18] N. K. Gill and S. Singh, "A Dynamic, Cost-Aware, Optimized Data Replication Strategy for Heterogeneous Cloud Data Centers," Future Generation Computer Systems, vol. 65, pp. 10-32, 2016.
[19] A. Gupta, R. Krishnaswamy, A. Kumar, and D. Panigrahi, "Online and Dynamic Algorithms for Set Cover," in Annual ACM SIGACT Symposium on Theory of Computing, Montreal, Canada, 2017, pp. 537-550.
[20] L. Trevisan, "Stanford University | CS261: Optimization," 2011.
[21] N. Buchbinder and J. Naor, "The Design of Competitive Online Algorithms via a Primal: Dual Approach," Foundations and Trends® in Theoretical Computer Science, vol. 3, pp. 93-263, 2009.
[22] Z. Wu, M. Butkiewicz, D. Perkins, E. Katz-Bassett, and H. V. Madhyastha, "SPANStore: Cost-Effective Geo-Replicated Storage Spanning Multiple Cloud Services," in ACM Symposium on Operating Systems Principles, Farminton, USA, 2013, pp. 292-308.
[23] The Top 20 Valuable Facebook Statistics – Updated May 2017. Available: https://zephoria.com/top-15-valuable-facebook-statistics/
[24] S. Knapton. (2016). Facebook Users Have 155 Friends - But Would Trust Just Four in a Crisis. Available: http://www.telegraph.co.uk/news/science/science-news/12108412/Facebook-users-have-155-friends-but-would-trust-just-four-in-a-crisis.html
[25] A. Chen. (2013). How Do You Find Insights Like Facebook's "7 Friends in 10 Days" to Grow Your Product Faster? Available: https://www.quora.com/How-do-you-find-insights-like-Facebooks-7-friends-in-10-days-to-grow-your-product-faster?page_id=2#!n=12
[26] J. Baer. (2012). The Social Habit – Is Our Facebook Addiction Ruinous. Available: http://www.convinceandconvert.com/social-media-research/the-social-habit-is-our-facebook-addiction-ruinous/
[27] Leading Countries based on Share of Facebook Users Worldwide as of May 2015. Available: http://www.statista.com/statitics/264838/countries-with-the-most-facebook-users/
[28] G. Liu, H. Shen, and H. Chandler, "Selective Data Replication for Online Social Networks with Distributed Datacenters," IEEE Transactions on Parallel and Distributed Systems, vol. 27, pp. 2377-2393, 2016.
[29] L. Jiao, T. Xu, J. Li, and X. Fu, "Latency-aware Data Partitioning for Geo-replicated Online Social Networks," in the Workshop on Posters and Demos Track (PDT'11), Lisbon, Portugal, 2011.
[30] L. Jiao, J. Li, T. Xu, W. Du, and X. Fu, "Optimizing Cost for Online Social Networks on Geo-Distributed Clouds," IEEE/ACM Transactions on Networking, vol. 24, pp. 99-112, 2016.
[31] D. A. Tran and T. Zhang, "S-PUT: An EA-based Framework for Socially Aware Data Partitioning," Computer Networks, vol. 75, pp. 504-518, 2014.
[32] Z. Wang, B. Li, L. Sun, W. Zhu, and S. Yang, "Dispersing Instant Social Video Service Across Multiple Clouds," IEEE Transactions on Parallel and Distributed Systems, vol. 27, pp. 735-747, 2016.
[33] J. M. Pujol, V. Erramilli, G. Siganos, X. Yang, N. Laoutaris, P. Chhabra, and P. Rodriguez, "The Little Engine(s) That Could: Scaling Online Social Networks," IEEE/ACM Transactions on Networking, vol. 20, pp. 1162-1175, 2012.
[34] J. Zhou, and J. Fan, "JPR: Exploring Joint Partitioning and Replication for Traffic Minimization in Online Social Networks," in IEEE Conference on Distributed Computing Systems (ICDCS), Atlanta, USA, 2017, pp. 1147-1156.
[35] L. Jiao, J. Lit, W. Du, and X. Fu, "Multi-Objective Data Placement for Multi-Cloud Socially Aware Services," in IEEE Conference on Computer Communication (INFOCOM), Toronto, Canada, 2014, pp. 28-36.
[36] Z. Ye, S. Li, and J. Zhou, "A Two-Layer Geo-Cloud Based Dynamic Replica Creation Strategy," Applied Mathematics & Information Sciences, vol. 8, pp. 431-440, 2014.
[37] S. Han, B. Kim, J. Han, K. Kim, and J. Song, "Adaptive Data Placement for Improving Performance of Online Social Network Services in a Multicloud Environment," Scientific Programming, vol. 2017, pp. 1-17, 2017.
[38] J.-W. Lin, C.-H. Chen, and J. M. Chang, "QoS-Aware Data Replication for Data-Intensive Applications in Cloud Computing Systems," IEEE Transactions on Cloud Computing, vol. 1, pp. 101-115, 2013.
[39] H. Khalajzadeh, D. Yuan, J. Grundy, and Y. Yang, "Improving Cloud-Based Online Social Network Data Placement and Replication," in IEEE International Conference on Cloud Computing (CLOUD), San Francisco, USA, 2016, pp. 678-685.

[40] H. Khalajzadeh, D. Yuan, J. Grundy, and Y. Yang, "Cost-Effective Social Network Data Placement and Replication Using Graph-Partitioning," in IEEE International Conference on Cognitive Computing (ICCC), Honolulu, USA 2017, pp. 64-71.
[41] S. Borst, V. Gupta, and A. Walid, "Distributed Caching Algorithms for Content Distribution Networks," in IEEE International Conference on Computer Communications (INFOCOM), San Diego, USA, 2010, pp. 1-9.
[42] J. Liu and B. Li, "A QoS-Based Joint Scheduling and Caching Algorithm for Multimedia Objects," World Wide Web, vol. 7, pp. 281-296, 2004.
[43] F. Chen, K. Guo, J. Lin, and T. L. Porta, "Intra-cloud Lightning: Building CDNs in the Cloud," in IEEE International Conference on Computer Communications (INFOCOM), Orlando, USA, 2012, pp. 433-441.
[44] S. Nikolaou, R. V. Renesse, and N. Schiper, "Proactive Cache Placement on Cooperative Client Caches for Online Social Networks," IEEE Transactions on Parallel and Distributed Systems, vol. 27, pp. 1174-1186, 2016.
[45] H. Hu, Y. Wen, T.-S. Chua, J. Huang, W. Zhu, and X. Li, "Joint Content Replication and Request Routing for Social Video Distribution Over Cloud CDN: A Community Clustering Method," IEEE Transactions on Circuits and Systems for Video Technology, vol. 26, pp. 1320-1333, 2016.
[46] M. Hu, J. Luo, Y. Wang, and B. Veeravalli, "Practical Resource Provisioning and Caching with Dynamic Resilience for Cloud-Based Content Distribution Networks," IEEE Transactions on Parallel and Distributed Systems, vol. 25, pp. 2169-2179, 2014.
[47] Y. Wu, C. Wu, B. Li, L. Zhang, Z. Li, and F. C. M. Lau, "Scaling Social Media Applications into Geo-Distributed Clouds," IEEE/ACM Transactions on Networking, vol. 23, pp. 689-702, 2015.
[48] Q. Xia, W. Liang, and Z. Xu, "The Operational Cost Minimization in Distributed Clouds via Community-Aware User Data Placements of Social Networks," Computer Networks, vol. 112, pp. 263-278, 2017.
[49] B. Yu and J. Pan, "Location-Aware Associated Data Placement for Geo-distributed Data-Intensive Applications," in IEEE Conference on Computer Communications (INFOCOM), Kowloon, Hong Kong, 2015, pp. 603-611.
[50] J. Zhoua, J. Fan, J. Jia, B. Cheng, and Z. Liu, "Optimizing Cost for Geo-distributed Storage Systems in Online Social Networks," Journal of Computational Science, vol 26, pp. 363-374, 2018.
[51] Y. Sverdlik. (December 2017). Facebook to Build Two More Massive Data Centers in Oregon. Available: http://www.datacenterknowledge.com/facebook/facebook-build-two-more-massive-data-centers-oregon
[52] AWS Global Infrastructure. Available: https://aws.amazon.com/about-aws/global-infrastructure/?tag=vig-20
[53] Amazon CloudFront Product Details – Amazon Web Services. Available: https://aws.amazon.com/cloudfront/details/
[54] AWS Greengrass - Amazon Web Services. Available: https://aws.amazon.com/greengrass/

## Appendix: Table of Notations

Table of notations for our static and dynamic strategies is shown below.

| Notation | Meaning |
|---|---|
| $A$ | Subset of the universe $U$ in dynamic set cover |
| $c$ | Updated number of connections |
| $Connections$ | Updated matrix of connections |
| $D_{ijk}$ | Updated delay matrix of user $i$ |
| $Datacenters$ | Updated set of datacenters in the cloud environment |
| $DC_{main_i}$ | Main datacenter for user $i$ |
| $Delay$ | Acceptable latency |
| $F$ | Maximum number of all users' friends |
| $FriendsNum_i$ | Number of user $i$'s friends |
| $L_{ij}$ | Updated delay of user $i$ accessing datacenter $j$ |
| $L'_{ijk}$ | Updated delay of friend $j$ of user $i$ accessing datacenter $k$ |
| $Latency_r(ts)$ | Latency for request $r$ in time period $ts$ |
| $m$ | Updated number of datacenters |
| $MinReplica$ | Minimum number of replicas |
| $n$ | Updated number of users |
| $P$ | Desirable percentile |
| $r$ | Total number of requests from all users |
| $ReplicaNum_i$ | Final number of replicas for user $i$ |
| $RequestCost_i(dt)$ | Total request cost for user $i$ |
| $RequestNum_{ij}(ts)$ | Number of requests from friend $j$ of user $i$ in time period $ts$ |
| $RT$ | Replica access table |
| $S$ | Solution space |
| $S_{ij}$ | Whether data of user $i$ is stored in datacenter $j$ or not |
| $S_{ts}$ | The updated solution at time period $ts$ |
| $StorageCost_i(ts)$ | Total storage cost of user $i$ during time period $ts$ |
| $StoredDataSize_i(ts)$ | Data size for user $i$ at the end of time period $ts$ |
| $TotalCost(\$)$ | Total cost for all users |
| $TransferCost_i(ts)$ | Total transfer cost for user $i$ during time period $ts$ |
| $ts$ | Time period |
| $U$ | Set of all possible solutions in greedy algorithm |
| $UnitRRequestPrice_j(ts)$ | Price for read request in datacenter $j$ in time period $ts$ |
| $UnitStoragePrice_j(ts)$ | Price for storing one GB of data at the end of time period $ts$ in datacenter $j$ |
| $UnitTransferPrice_j(ts)$ | Price for transferring one Gigabyte of data from datacenter $j$ in time period $ts$ |
| $UnitWRequestPrice_j(ts)$ | Price for write request in datacenter $j$ in time period $ts$ |
| $UpdateCost_i(ts)$ | The cost for synchronizing user $i$'s replicas from the primary replica at the end of time period $ts$ |
| $Users$ | Updated list of the users in the system |